PSis - Programação de Sistemas - 2017/2018
Exame 1ª Época, 14 de Junho de 2018, 18h30, Duração: 2h

Software Architectures and Testing [7.5]

1. [1.0 points] What is the **architecture of a system**?

It is the description of the various components and how they relate.
Model to describe/analyze a system
Structure of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them

2. [1.5 points] The project (distributed clipboard) implemented two different **deployment architecture patterns**.
Describe those two architectural patterns in the context of the project

Client/Server – Aplicação → clipboard
N-Tier / 3-Tier – várias camadas Aplicação → API → servidor → regioes
Peer-to-peer – Vários clipboards

3. [1.5 points] Describe the principle of **Least Knowledge.**
Describe how it was applied in the project.
What are the advantages of its use?

A component or object should not know about internal details of other components or objects.
The API hide completely how the clipboard was implemented (Remote/local)
It is possible to change the implementation of the API without change the applications

4.a. [0,5 points] What is **validation testing**? What class of errors this tests try to find?

Tests that demonstrate agreement with requirements.
Tries to fins deviations between the implementation and the defined requirements:

- functional requirements are satisfied,
- all behavioral characteristics are achieved,
- all performance requirements are attained,
- documentation is correct,
- other requirements are met

4.b. [0,5 points] Describe the main difference between **beta and alpha testing**

Alpha tests are conducted in a controlled environment (at the developers site)
Beta tests are conducted in real environment (at the customer site)

Alpha - errors are recorder by developer
Beta - errors are reported by the customer

5. In the project, the 10 regions of each clipboard could be implemented as a module with a clearly defined API.
5.a. [1.0 points] Describe the various f**unctions/procedures** of that API by writing the declaration or prototypes of those functions.

```
Reg * init_regions()
int store_data(reg * r, int reg_id, void * data, int length)
int get_data(reg * r, int reg_id, void * * data, int *length)
```

5.b. [1.5 points] Describe **3 unitary tests** that could be applied to that module, and present for each test the **error that would be found**.

```
Retrive data from an empty region
Retrieve/store data form an invalid region (reg_id large)
Store large data
```

Operating Systems [3.0]

7. [1.0 points] Describe the **two types of portability** that are offered by current operating systems.

Portability of application that run on different operatying systems
Portability of the OS that runs on multiple HW

8. [1.0 points] Describe the fundamental **mechanism** that allow the easy implementation of **portability**.

Abstraction is fundamental to obtains this portability. A component or layer hide completely the implementation details allowing it to be ported/ implemented in different systems

9. [1.0 points] Describe why **security** became a concern when operating systems started being **time-shared**.
What is the **architectural pattern** that allows the easy implementation of the security mechanisms?

Layers allow the implemenation of abstraction and are also implemented by the processors

Processes / Threads / IPC / Shared memory [4.0]

10. [1.0 point] Can **signals** be considered **inter-process communication** mechanisms? Justify.

Yes, they can be used to inform aother aplication of specialk event of “order”other application to execute some code

11. In the project, the transfer of the data between the application and the local clipboard was performed using sockets, but shared memory could also be used.
11.a. [0.5 points] What would be the **advantages of using shared memory**?

The transfer could be faster since it would not be necessary to copy data to the kernel and back to the other processe

11.b. [0.5 points] What type of **issues** should be taken into consideration when programming communication using shared memory?

The mains issue that must be take into consideration is the sychronization on the access to the memory. Furthemore the address space is different between processes and that must be taken into consideration

11.c. [1.0 points] Write the **pseudo-code of the clipboard_copy** function using **shared memory**.

```
Int clipboard_copy (fd, region, data, len){
     prt _ new = create_shared_memory_region(shmem_id, data)
     memcpy(prt_new, data, len)
     send(fd, "copy_shm")
     send(fd, region)
     send(fd, shmem_id)
}
```

12. [1.0 point] Explain how the UNIX/LINUX communication **API** implements **data abstraction** with respect to the c**ommunication channels**.

All communication channels are represented by an integer. The programmer does not know what its type is nor the implementation of the protocol. The same set of functions work for socket, pipes, and files.

Synchronization [5.5]

13. In the project, it was necessary to forward information regarding copies to other clipboards.
This could be performed iterating over a list of sockets in multiple threads, as presented in the next pseudo-code.

```
for (aux = clipboard_list; aux != NULL, aux = aux→seg){
   send(aux->sock_fd, msg, sizeof(msg));
   send(aux→sock_fd, data, len_data);
   if( error in send)
      remove(clipboard_list, aux)
}
```

13.a. [1.0 points] Describe and explain **two issues** that can occur when the previous code is executed **simultaneously in different threads**.

The two send can be interleaved by a send to the same clipboard but by a different thread
the remove can free a list stucture that wil be accessed by another thread

13.b. [1.0 points] Correct the **previous issues** by adding the suitable **synchronization** to the code.

```
lock(m1)
for (aux = clipboard_list; aux != NULL, aux = aux→seg){
   send(aux->sock_fd, msg, sizeof(msg));
   send(aux→sock_fd, data, len_data);
   if( error in send)
      remove(clipboard_list, aux)
}
unlock(m1)
```

14. [1.0 point] Although mutexes can be replaced by semaphores that should not be done.
Explain why.

The mutexes offer aditional security measures to guarantee that the regions are well guardedavoid deadlocks:

- unlock by another thread
- dead of the owner of the lock
- mutiple locks by the same thread

15. [2.5 points] One way to perform communication between multiple threads is using shared variables. The following code allows the communication between threads (consumers and producers) using an array of 2 positions (**v**) that is filled by the producers and read by the consumers:

- The array **v** starts empty (both positions NULL)
- There are multiple producers (Thread A) and consumers (Thread B) running at the same time
- Producers should only access **v** if it is possible to store information on an empty slot
- Consumer should only access **v** if it is possible to read information from a filled slot
- Each piece of data written in the array **v** should be processed by a single consumer

```
void * v[2] = {NULL, NULL};
```

Thread A - Producer

```
while (1){
   p = produceValue()
   //wait for available space on v
   if(v[0] == NULL)
      v[0] = p;
   else
      v[1] = p;
}
```

Thread B - Consumer

```
while(1){
    //wait for data on v
    if(v[0] != NULL)
      processeValue(v[0]);
      free(v[0])
      v[0] = NULL;
   else
      processeValue(v[1]);
      free(v[1])
      v[1] = NULL;
}
```

Define the necessary synchronization variables, initialize them correctly and change the code to guarantee that:

- It works correctly (no data is lost and is processed by one consumer)
- Solves all race conditions
- Critical regions are the smallest possible

```
void * v[2] = {NULL, NULL};
sem_cons = 0, sem_prod = 2; mux_v
```

Thread A - Producer

```
while (1){
   p = produceValue()

   wait(sem_prod)

   lock(mux_v);

   if(v[0] == NULL)
      v[0] = p;
   else
      v[1] = p;

   unlock(mux_v)

   signal(sem_cons)
}
```

Thread B - Consumer

```
while(1){
    wait(sem_cons)

    lock(mux_v);
    if(v[0] != NULL)
      aux = v[0];
      v[0] = NULL;
   else
      aux = v[1];
      v[1] = NULL;
   unlock(mux_v)

   signal(sem_prod)

   processValue(aux)
   free(aux)
}
```